Import necessary libraries

In [1]:
```
import numpy as np
from sklearn import datasets, tree
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

In [2]:
```
digits = datasets.load_digits()
```

Load the digits dataset

In [5]:
```
digits = datasets.load_digits()
```

In [6]:
```
digits
```

```
  'pixel_6_7',
  'pixel_7_0',
  'pixel_7_1',
  'pixel_7_2',
  'pixel_7_3',
  'pixel_7_4',
  'pixel_7_5',
  'pixel_7_6',
  'pixel_7_7'],
 'target_names': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 'images': array([[[ 0.,  0.,  5., ...,  1.,  0.,  0.],
         [ 0.,  0., 13., ..., 15.,  5.,  0.],
         [ 0.,  3., 15., ..., 11.,  8.,  0.],
         ...,
         [ 0.,  4., 11., ..., 12.,  7.,  0.],
         [ 0.,  2., 14., ..., 12.,  0.,  0.],
         [ 0.,  0.,  6., ...,  0.,  0.,  0.]],

        [[ 0.,  0.,  0., ...,  5.,  0.,  0.],
         [ 0.,  0.,  0., ...,  9.,  0.,  0.],
```

In [7]:
```
digits.feature_names
```

```
 'pixel_2_7',
 'pixel_3_0',
 'pixel_3_1',
 'pixel_3_2',
 'pixel_3_3',
 'pixel_3_4',
 'pixel_3_5',
 'pixel_3_6',
 'pixel_3_7',
 'pixel_4_0',
 'pixel_4_1',
 'pixel_4_2',
 'pixel_4_3',
 'pixel_4_4',
 'pixel_4_5',
 'pixel_4_6',
 'pixel_4_7',
 'pixel_5_0',
 'pixel_5_1',
 'pixel_5_2',
```

In [8]:
```
digits.target_names
```

Out[8]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [9]:
```
digits.data
```

Out[9]:
```
array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]])
```

In [10]:
```
# Inspect dataset attributes
# The digits dataset contains pixel data for handwritten digits (0-9)

print(digits.feature_names)  # Feature names
print(digits.target_names)   # Target class names (digits 0-9)
```

```
['pixel_0_0', 'pixel_0_1', 'pixel_0_2', 'pixel_0_3', 'pixel_0_4', 'pixel_0_5', 'pixel_0_6', 'pi
xel_0_7', 'pixel_1_0', 'pixel_1_1', 'pixel_1_2', 'pixel_1_3', 'pixel_1_4', 'pixel_1_5', 'pixel_
1_6', 'pixel_1_7', 'pixel_2_0', 'pixel_2_1', 'pixel_2_2', 'pixel_2_3', 'pixel_2_4', 'pixel_2_
5', 'pixel_2_6', 'pixel_2_7', 'pixel_3_0', 'pixel_3_1', 'pixel_3_2', 'pixel_3_3', 'pixel_3_4',
'pixel_3_5', 'pixel_3_6', 'pixel_3_7', 'pixel_4_0', 'pixel_4_1', 'pixel_4_2', 'pixel_4_3', 'pix
el_4_4', 'pixel_4_5', 'pixel_4_6', 'pixel_4_7', 'pixel_5_0', 'pixel_5_1', 'pixel_5_2', 'pixel_5
_3', 'pixel_5_4', 'pixel_5_5', 'pixel_5_6', 'pixel_5_7', 'pixel_6_0', 'pixel_6_1', 'pixel_6_2',
'pixel_6_3', 'pixel_6_4', 'pixel_6_5', 'pixel_6_6', 'pixel_6_7', 'pixel_7_0', 'pixel_7_1', 'pix
el_7_2', 'pixel_7_3', 'pixel_7_4', 'pixel_7_5', 'pixel_7_6', 'pixel_7_7']
[0 1 2 3 4 5 6 7 8 9]
```

Features (pixel data) and target labels

In [12]:
```
X = digits.data            # Pixel data
Y = digits.target          # Corresponding digit labels
```

In [13]:
```
X
```

Out[13]:
```
array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]])
```

In [14]:
```
Y
```

Out[14]:
```
array([0, 1, 2, ..., 8, 9, 8])
```

In [15]:
```
# Number of samples in the dataset
number_of_samples = len(Y)
print(f"Number of samples: {number_of_samples}")
```

```
Number of samples: 1797
```

Splitting the dataset into training, validation, and test sets

70% training, 15% validation, 15% testing

--

In [16]:
```
# Shuffle the dataset with a random permutation
random_indices = np.random.permutation(number_of_samples)
```

In [17]:
```
# Training set (70%)
num_training_samples = int(number_of_samples * 0.7)
x_train = X[random_indices[:num_training_samples]]
y_train = Y[random_indices[:num_training_samples]]
```

In [18]:
```
# Validation set (15%)
num_val_samples = int(number_of_samples * 0.15)
x_val = X[random_indices[num_training_samples:num_training_samples + num_val_samples]]
y_val = Y[random_indices[num_training_samples:num_training_samples + num_val_samples]]
```

In [19]:
```
# Test set (15%)
num_test_samples = int(number_of_samples * 0.15)
x_test = X[random_indices[-num_test_samples:]]
y_test = Y[random_indices[-num_test_samples:]]
```

--

Decision Tree Classifier using Entropy criterion

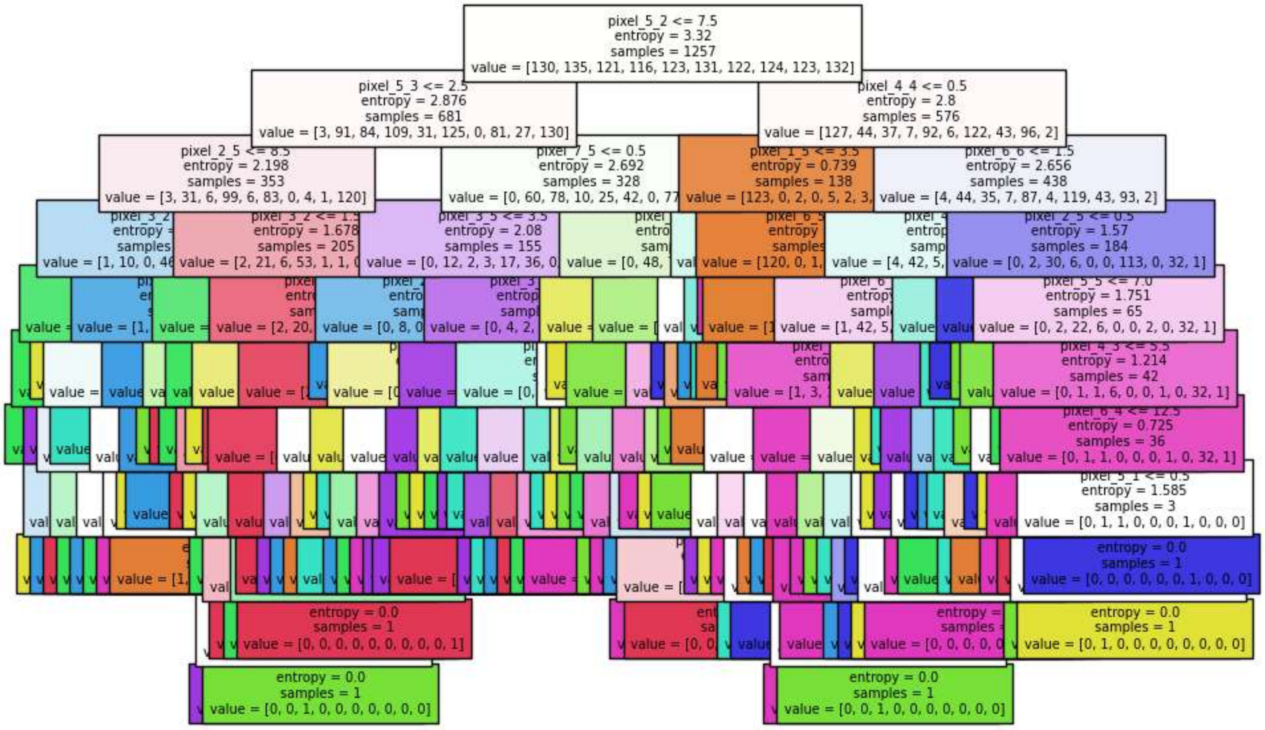
--

In [20]:
```
# create model
model_entropy = tree.DecisionTreeClassifier(criterion="entropy", max_depth=15)
model_entropy.fit(x_train, y_train)
```

Out[20]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=15)
```

Visualize the decision tree

In [21]:
```
plt.figure(figsize=(10, 7))
tree.plot_tree(model_entropy, filled=True, fontsize=7, feature_names=digits.feature_names)
plt.show()
```



Evaluate

In [24]:
```
# ----------------------------------
# Validate the model on the validation set
# ----------------------------------
validation_predictions = model_entropy.predict(x_val)
```

In [25]:
```
# Calculate misclassifications on validation set
val_misclassifications = np.sum(validation_predictions != y_val)
print(f"Validation Misclassifications: {val_misclassifications}")
```

```
Validation Misclassifications: 34
```

In [26]:
```
# Show misclassified examples (predicted vs actual)
misclassified = np.concatenate((validation_predictions.reshape(-1, 1), y_val.reshape(-1, 1)), ax
print("Misclassified Validation Samples (Prediction, Actual):")
print(misclassified[validation_predictions != y_val])
```

```
Misclassified Validation Samples (Prediction, Actual):
[[1 9]
 [6 2]
 [8 2]
 [9 6]
 [8 7]
 [0 6]
 [6 3]
 [4 7]
 [2 3]
 [3 9]
 [5 3]
 [3 7]
 [9 5]
 [5 1]
 [3 0]
 [1 9]
 [5 7]
 [7 2]
 [2 3]
 [6 4]
 [3 9]
 [3 9]
 [9 2]
 [8 7]
 [0 4]
 [3 9]
 [1 9]
 [4 7]
 [4 9]
 [7 4]
 [9 8]
 [5 3]
 [8 3]
 [6 4]]
```

In [27]:
```
# Misclassification percentage on validation set
validation_misclassification_percentage = val_misclassifications * 100 / len(validation_predicti
print(f"Validation Misclassification Percentage: {validation_misclassification_percentage:.2f}%"
```

```
Validation Misclassification Percentage: 12.64%
```

In [28]:
```
# ----------------------------------
# Test the model on the test set
# ----------------------------------
test_preds = model_entropy.predict(x_test)
```

In [29]:
```
# Calculate misclassifications on test set
test_misclassifications = np.sum(test_preds != y_test)
print(f"Test Misclassifications: {test_misclassifications}")
```

```
Test Misclassifications: 45
```

In [30]:
```
# Misclassification percentage on the test set
test_misclassification_percentage = test_misclassifications * 100 / len(test_preds)
print(f"Test Misclassification Percentage: {test_misclassification_percentage:.2f}%")
```

```
Test Misclassification Percentage: 16.73%
```

Now applying decision tree to a different dataset (Iris dataset)

Load Iris dataset

In [31]:
```
iris = datasets.load_iris()
```

In [32]:
```
print(iris.feature_names)   # Features of Iris dataset
print(iris.target_names)    # Target class names (Setosa, Versicolor, Virginica)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
['setosa' 'versicolor' 'virginica']
```

In [33]:
```
# Features (sepal & petal dimensions) and target labels (species)
X = iris.data
Y = iris.target
```

Split Iris data into training and testing sets (70% training, 30% testing)

In [35]:
```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=3)
```

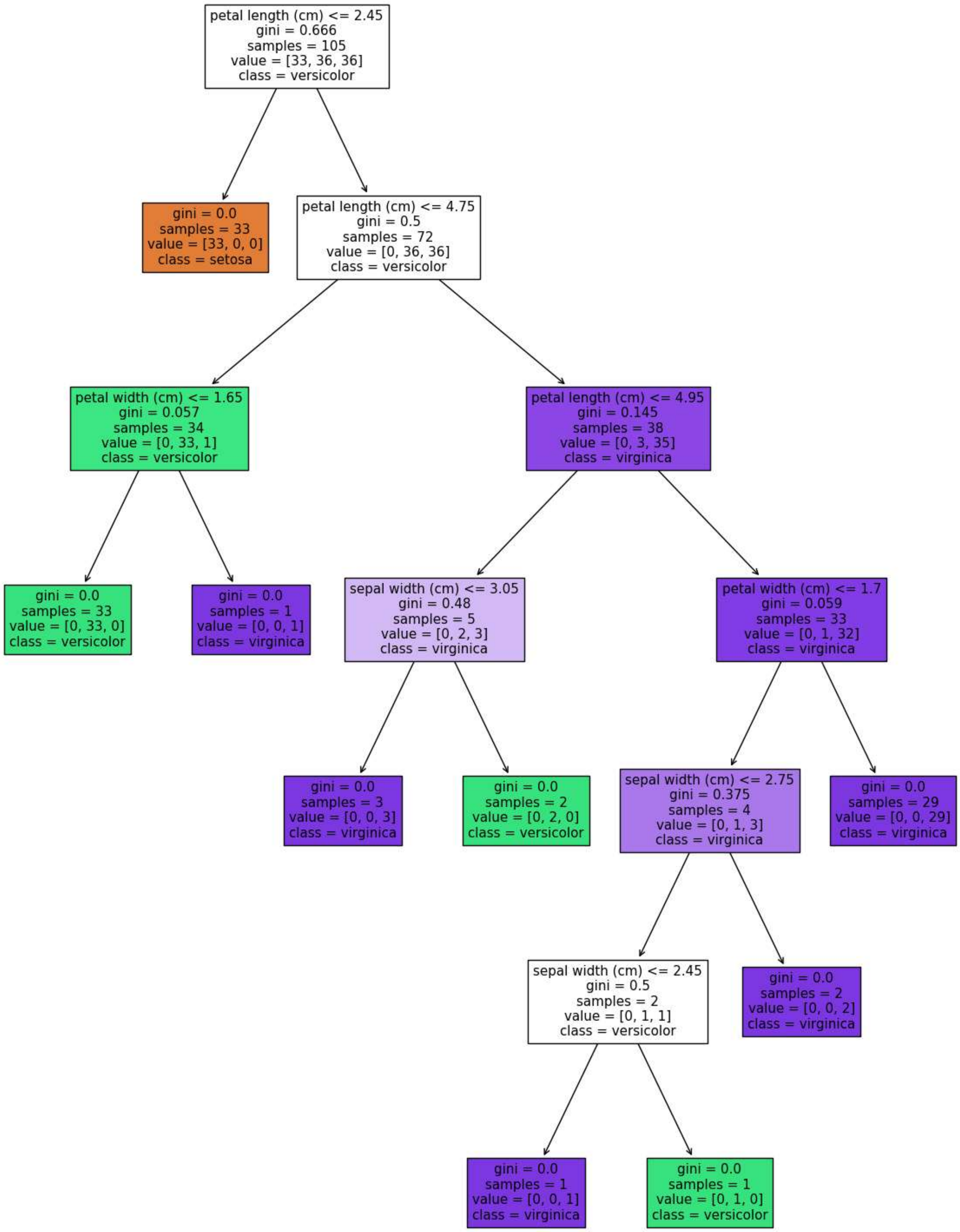
Model and train

In [36]:
```
# Train Decision Tree model on Iris dataset
clf = tree.DecisionTreeClassifier()
clf.fit(x_train, y_train)
```

Out[36]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

Visualize

In [38]:
```
# Visualize the decision tree for the Iris dataset
plt.figure(figsize=(15, 20))
tree.plot_tree(clf, feature_names=iris.feature_names, class_names=iris.target_names, filled=True
plt.show()
```



In [39]:
```
# Predict the test set results for Iris
test_preds = clf.predict(x_test)
```

In [40]:
```
# Compare predictions with actual test set labels
print(f"Predictions: {test_preds}")
print(f"Actual Labels: {y_test}")
```

```
Predictions: [0 0 0 0 0 2 1 0 2 1 1 0 1 1 2 0 2 2 2 0 2 2 2 2 0 2 1 1 1 1 0 0 2 1 0 0 2
 0 2 1 2 1 0 0 2]
Actual Labels: [0 0 0 0 0 2 1 0 2 1 1 0 1 1 2 0 1 2 2 0 2 2 2 1 0 2 2 1 1 1 0 0 2 1 0 0 1
 0 2 1 2 1 0 0 2]
```

In [41]:
```
# Calculate misclassifications on test set
test_misclassifications = np.sum(test_preds != y_test)
print(f"Test Misclassifications (Iris): {test_misclassifications}")
```

```
Test Misclassifications (Iris): 4
```

In [42]:
```
# Misclassification percentage on the test set
test_misclassification_percentage = test_misclassifications * 100 / len(test_preds)
print(f"Test Misclassification Percentage (Iris): {test_misclassification_percentage:.2f}%")
```

```
Test Misclassification Percentage (Iris): 8.89%
```